Jamal Blackwell

Course 6

Location: Media Lab

Building a Video-centered web community

My UROP consisted of assisting Pengkai Pan with designing and implementing a portion of the software and web pages necessary to construct a web community centered on video content. In such a community, people would be able to communicate with other members of the community to discuss the videos on the site and the variations on those videos that the users have produced. Further, sharing this video content with others is facilitated through the use of emailing the URL of the clip to the email address of the viewer. In building this community, our goal was to get away from the current model of how video is shared. Instead of a single author producing one "definitive" arrangement of clips into final project, we hoped to allow multiple authors to collaborate remotely on many arrangements of various video clips. For this task, we will draw heavily on the work done by Penkai Pan on the I-views project.

In I-views, Penkai Pan constructed an online site that would allow people to form a web community centered on video content. Users were able to edit and append video clips together to produce new video sequences for viewing by other users. There were tools that measured the similarity of video clips based on the number of common sources the two video clips had. A user could then contact a user who had produced a very similar or dissimilar video from the user's own video via email. In this way I-views facilitated discussion among the members of its community. In our new project, we wished to make joining the community more accessible by simplifying the user interface and providing new and varied activities for our users' enjoyment.

In order to build such a community, we had to give people an incentive to join the site, recommend it to friends, and interact with other members of the community. To achieve this task, I was to design and implement various entertaining activities for the users to engage in during their visits to our site. Some activities were to be more game oriented, while others could have focused on interactive story telling by multiple users. For instance, one possible game idea would have involved people providing the narration for a sequence of video arranged by another user. All the game's participants could then give each other a score based on the overall quality of the narration. This game illustrates our main focus when we designed these activities, in that we entertain the members of the community while strongly encouraging multiple user interaction. It was our belief that users entertained by our site would be prone to refer it to friends and remain active in our video-based community.

In order to get people to return to the site repeatedly, we will needed to have new activities for them to perform. For this reason, video editing will be incorporated into many activities. In this way, we allowed part of our user's entertainment stem from providing content for other users in the community. Much like friends sitting around a campfire, our users would have the ability to tell new stories to each other and to alter the previously told stories in a more entertaining manner. Further, all games featured on the site were to have special templates upon which they are constructed. These templates were to be available for users to construct their own versions of the games we provide. This is another way in which users may provide entertainment for each other. Finally, conventional chat is supported on the site in an effort to encourage user-to-user interaction within the community. One might combine all of these aspects into a lobby where one can watch video, edit videos, edit game, or participate in games, while chatting with their fellow community members. We believe such an interactive and rewarding experience would keep users coming back and cause our community to continually grow.

In order for users to share and author video content, one would think that they would need a digital video camcorder and video editing software. Video editing tools or Digital Video (DV) camcorders are expensive and typically owned by web users. Therefore, most of our users needed an alternative means of video production, lest they feel unable to contribute to the community. The use of SMIL files allowed us to link together various video clips on the web, without the use of expensive equipment. We saved storage space by allowing users to "add" content to the community's video store by submitting the URL of the clip instead of the actual video. URLs can be included in SMIL playlists and thus a web address was enough to allow us to share a clip with the entire community. This is the same solution used in I-views. We wish to automatically generate these SMIL files so that the user's editing experience will mainly consist of juxtaposing video clips via a graphical interface. Another UROP was assigned this task, so I will refer to the SMIL editor as an abstraction that performs gives the users video editing capability. Our main goal was to make joining and participating in the community as enjoyable as possible, thus simplifying complex tasks will reduce the learning curve associated with video editing techniques.

Available technology served to determine my choice of activities to include on the site. More ambitious game ideas were scrapped due to the most conventional video media player's lack interactivity with the users. These tools, such as the Quicktime movie viewer and Realplayer were built with the old model of authorship in mind, in which one person is simply putting his video on display for others to view. Presumably this was the impetus for these player's restricting the forms of user interaction to those similar to a common VCR. Functionality such as recognizing the area within a video clip that a user clicks on and associating new events with certain keystrokes were not useful under this broadcast model, but would have formed the basis for my more ambitious ideas. However, all hope was not lost by virtue of the SMIL editing applet. The ability to allow users to produce their own unique video sequences can serve to provide the large amount of video content necessary to support some of my other ideas.

The front-runner among my technically feasible projects was founded on mimicking a common real life situation. I considered the following scenario when I came up with the idea for the x-views viewing room. Suppose there is a championship sporting event being broadcast on television; lets say it is the Superbowl for the sake of specificity. I call my friends and we talk while watching the game, commenting on the plays as they happen. Finally, half-time arrives and we are not satisfied with the entertainment provided, so we start to change channels in search of something more compelling to watch. While independently searching for entertainment we continue talking and updating each other on our progress. With this in mind the structure for the viewing room became clear, users should be able to communicate with each other regardless of the video clip they are watching. They should be able to provide each other with advice on where to search for entertainment and comment on the clips they enjoyed and those they did not enjoy. This is the functionality I sought to provide users in the online viewing room.

Viewing the clips in the repository should be as dynamic as the conversations about them are likely to be. With this in mind, I thought it best to use keyword searching and assignment to gauge user's thoughts on a given clip. After or during the viewing of a clip, users would assign a clip a list of predefined keywords and submit them to the database. It is the general consensus that qualitative evaluations of this kind are more meaningful in the realm of short video clips and a quantitative value, such as "2 thumbs up". The user would then be able to select a keyword categorizing the type of clip she would like to view. This selection would cause the drop-down menu to p. If so, the drop-down selection menu should be populated with the clips in this category and the user would be free to chose from among these clips which one she would see next. All the while, the user is interacting with other users enjoying the same activity that she is.

The overall structure of the viewing room is fairly simple. Two web pages are used in frames to create the viewing area, a video display frame and a chat frame. The top-frame is a web page featuring little more than an embedded Realplayer. Frames were used due to the Realplayer's inability to loop a video clip, so it is necessary to refresh the page at the end of a clip's running time in order to create this effect. The real format was chosen since streaming video is readily available in this format. Some people have asked why the realplayer was chosen and not a quicktime player. Both players are SMIL, MPEG, and AVI compatible and support their own unique streaming media formats, with the quicktime movie viewer boasting a built-in looping feature. In my estimation, the advantage of built-in looping is minimal and offset by the real server being free while the streaming quicktime server is available at a fee. This make is more likely to be used by smaller web developers who simply want to feature video on their personal web sites. Any users can easily add the URLs of these clips to our repository and edit them in the sequencing applet. In short, it seemed that this choice was, at the very least, as efficacious as embedding a Quicktime player in the video display frame. Further, it would be a trivial matter to replace the realplayer with a quicktime player at a later date.

It is interesting to consider how well the viewing room compares to the scenario that inspired its creation. The inherent differences in the nature of the communication methods involved, namely chat and verbal communication, are not to be overlooked. It might have been a far better model of the scenario had the chatroom employed telephony software, verbal Internet chat, to facilitate user-user interactions. I opted against this since the software is not free or ubiquitous. It requires speaker and a microphone, in addition to the phone emulation software. Having a comparably obscure and costly communication method would limit our user base and make using our site prohibitive to some people. Further, viewing online videos is a bandwidth intensive activity and to accommodate those with a slower Internet connection, chat is the preferred option.

The use of chat comes with the added bonus of allowing users to easily adjust the amount of attention they pay to the conversation and the video. Voice would compete with the sound in the video clips and perhaps lessen the experience viewing some clips. Further, it is not apparent that most computer systems would be able to handle simultaneous sound output from the realplayer and the telephony software. For this reason, verbal communication might imply the total removal of sound from the clips in our repository. My choice of chat in this application comes at the cost of limiting the emotion that can be conveyed and making the communication less intuitive for our users, but I thought the pros outweighed the cons in this situation.

The chat applet is my most useful and versatile contribution to x-views. I started to build my own chat applet, but mid-way through development I decided to search the Java Repository for a similar application. The Repository is known for having a wide variety of open source applications that tend to be useful and well written. I decided to take a promising program called JavaTalk for a test run. Hoping to find some answers on some of the trickier issues I encountered while developing my own applet, I instead found that JavaTalk had all of the functionality I wanted to add to our site. My questions about how to handle and create user sessions, connection and the notion of rooms were all dealt with, in addition to features that give the application a quite professional feel. Users can "whisper" to each other by highlighting the username of another chat participant and send a message that will only appear on the screen of the specified user. User blocking is also available to allow users to ignore the input of individuals that seek to annoy other chat participants. This allows us to truly let an autonomous community form, without the need for a chat room administrator to mediate every conversation.

There is no notion of rooms in JavaTalk, but instead only one chat is associated with each instance of the JavaTalk Server running on the web server. Users will not have to fumble around an overwhelming list of chatrooms, as is the case with most IRC servers that I have encountered. Relevant prompts and information, such as JavaTalk's emote commands, are detailed in the client applet, eliminating the need for users to refer to external documentation. Since the point of x-views is to build a video-centered web community, having only a select few chats hosted on the site makes interaction between a wide variety of different users much more likely. Further, novice chat users will not be intimidated by the vast number of chat options available to them. I think this the success of AOL tells us a valuable lesson: simple is sometimes the most popular option.

The administration of the JavaTalk Server is likewise simpler than that of a complex IRC server. To add more than one chat to x-views running another instance of the JavaTalk Server associated with another web page containing the client applet is necessary. Since the computational demands of JavaTalk are not extensive, having multiple chat servers running on the main x-views server is not likely to bog down the system. If it was necessary to shut down a chat, one could just shut down the associated server (identified by the port with which it is associated) and continue to let the other chats proceed uninterrupted. Creating a chat room is as simple as running the JavaTalk server using any version of the Java Development Kit (JDK) greater or equal to 1.1, with a number as its argument. The one parameter is to specify the port that the server is associated with. A page can then be created in which the client applet is embedded and associated with a port number. The JavaTalk package comes with a pre-written HTML page which handles most of the work. All one needs to do is edit the existing page and change the port parameter to equal the port number of its associated server. More options are available to the advanced administrator, such as idle time before a user is automatically logged out and log file name, and are described in the attached documentation of the JavaTalk package.

The uses for JavaTalk, or just online chat in general, are quite varied and afford great depth to x-views. As an applet, JavaTalk can be placed in a frame adjacent to any other x-views page and provide chat throughout the site. This would go a long way towards building and sustaining an online community. Further, it is a small, simple to use applet which makes it ideal for a site that makes excessive bandwidth demands in other ways, such as by providing streaming video. Without having to support IRC servers or other more standardized chat protocols, we simplify and streamline the user's experience. All users need to do in order to chat is to point a java-enabled web browser to the web page holding the applet, type in their desired username and begin chatting. Had we supported IRC we would have to run an IRC server and support numerous features such as "whois" and numerous "emote" commands that are not worth the extra development time to incorporate. We simply want to connect the users to each other for the time being, with features to come mainly due to user demand.

JavaTalk was more than adequate for our purposes in most respects, but did fall short in one area, namely its logging feature. The log file contains only data on how many users logged in on a given day. It also shows data of how many messages a user has sent successfully. For some reason, I can not get the log file to store the complete conversation logs for a given server. The code seems to specify that it should, but the bug may have something to do with the original JavaTalk package being written 3 years ago and run on a much later version of the JDK. This feature would allow us to store and associate each user with the conversations he has on our site, thus providing a history feature similar to that of ICQ.

The various prototypes of the viewing room seemed to indicate that the design and implementation were sound. I the prototype from my computer serving a real video in the upper frame of the page and found that the site loaded quickly and neither application was noticeably hindered by inhabiting the same browser. The final version of the viewing room was not built due to the API that will allow the video repository to communicate with the other components of x-views not being completed as of May 22, 2000. Any integration that could have been done would have been purely cosmetic as the state of the API was such that the viewing room would have to know the universal identification number of the clips it would allow the users to browse. Therefore, the viewing room would have to be rewritten every time that the database of clips was updated. Although the final version of the viewing room was not completed, this document provides an adequate blueprint for its construction. I believe that it would only take another week of development for someone to learn the x-views API and use it to assemble the viewing room from the components I specified.

JavaTalkServer Source Code

```
// $RCSfile: JavaTalkServer.java,v $
//
// $Id: JavaTalkServer.java,v 1.17 1996/06/27 23:43:21 fms Exp fms $
//
// by Frank Stajano, http://www.cam-orl.co.uk/~fms
//
// Development started on 1996 05 09
// (c) Olivetti Research Limited

/*
Copyright (c) 1996 Olivetti Research Limited

Permission is hereby granted, without written agreement and
without license or royalty fees, to use, copy, modify, and
distribute this software and its documentation for any purpose,
provided that the above copyright notice and the following three
paragraphs appear in all copies of this software, its documentation
and any derivative work.

In no event shall Olivetti Research Limited be liable to any party
for direct, indirect, special, incidental, or consequential damages
arising out of the use of this software and its documentation, even
if Olivetti Research Limited has been advised of the possibility of
such damage.

Olivetti Research Limited specifically disclaims any warranties,
including, but not limited to, the implied warranties of
merchantability and fitness for a particular purpose. The
software provided hereunder is on an "as is" basis, and Olivetti
Research Limited has no obligation to provide maintenance,
support, updates, enhancements, or modifications.

Derived or altered versions must be plainly marked as such, and
must not be misrepresented as being the original software.
*/


// TODO:
// make various logging activities be methods of the applet, not the
threads
// make broadcast be a method of the applet
// remove debugging printouts
// check if access to participants shouldn't be synchronized
// tell others whether the one who left did so by choice or was kicked
out

// ADVANCED TODO
// remember names of participants based on address (1st guess at least)
// implement timestamp echo protocol
// synchronise clocks and compute half-paths
// non-positional handling of command-line arguments


import java.io.*;
import java.net.*;
import java.util.*;

public class JavaTalkServer extends Thread implements Trigger {
// These are automatically substituted
public static final String rcsVersion = "$Revision: 1.17 $";
public static final String rcsDate = "$Date: 1996/06/27 23:43:21 $";
public static final String rcsProgram = "$RCSfile:
JavaTalkServer.java,v $";

// These extract the useful parts from the automatically substituted
ones.
public static final String version = rcsVersion.substring(11,
rcsVersion.length()-2);
public static final String date = rcsDate.substring(7, 17);
public static final String program = rcsProgram.substring(10,
rcsProgram.length()-9);
public static final String about = "";
public final static int DEFAULT_PORT = 6764;
public final static int WARNING_IF_INACTIVE_FOR = 5*60;
public final static int TIME_TO_LIVE_AFTER_WARNING = 60;
// seconds before a connection is closed
// (later transformed into milliseconds)
public final static int REAPER_PERIOD = 10*1000;
// milliseconds between checks of who is inactive

protected int warningIfInactiveFor = WARNING_IF_INACTIVE_FOR;
protected int timeToLiveAfterWarning = TIME_TO_LIVE_AFTER_WARNING;
protected int port = DEFAULT_PORT;
protected String logFileName = "javatalk." + Time.ymdhms(false) +
".log";

protected int maxInactivity;
protected PrintStream log;
protected ServerSocket listen_socket;
protected Hashtable participants;
protected Ticker ticker;

public static void main(String[] args) {
//System.err.println("main() called");
System.out.println(about);

new JavaTalkServer(args);
}

public static void usage() {
System.out.println ("USAGE (all arguments optional and positional):\n"
+ " java " + program
+ " warningIfInactiveFor timeToLiveAfterWarning port logFileName\n"
+ " warningIfInactiveFor After this many seconds of inactivity:
warning\n"
+ " timeToLiveAfterWarning After this many extra seconds: kicked out\n"
+ " port Port number on which to listen for connections\n"
+ " logFileName A transcript of the talk goes here\n"
);
System.exit(1);
}
```

```
public JavaTalkServer(String[] args) {

// parse arguments
switch (args.length) {
case 4:
logFileName = args[3];
// fallthrough
case 3:
try {
port = Integer.parseInt(args[2]);
}
catch (NumberFormatException e) {
usage();
}
// fallthrough
case 2:
try {
timeToLiveAfterWarning = Integer.parseInt(args[1]);
}
catch (NumberFormatException e) {
usage();
}
// fallthrough
case 1:
try {
warningIfInactiveFor = Integer.parseInt(args[0]);
}
catch (NumberFormatException e) {
usage();
}
// fallthrough
case 0:
break;

default:
usage();
}


// Print out the actual parameters for this invocation
System.out.println (
"\nwarningIfInactiveFor " + warningIfInactiveFor
+ "\ntimeToLiveAfterWarning " + timeToLiveAfterWarning
+ "\nport " + port
+ "\nlogFileName " + logFileName
+ "\n\nServer running and waiting for connections..."
);

// transform seconds into milliseconds, because that's what
// the Java library routines eat.
warningIfInactiveFor *= 1000;
timeToLiveAfterWarning *= 1000;

maxInactivity = warningIfInactiveFor + timeToLiveAfterWarning;

try {
```

```
log = new PrintStream(new FileOutputStream(logFileName));
}
catch (IOException e) {
fail(e, "Can't open log file '" + logFileName + "' for writing.");
}
log.println(about);
Date d = new Date();
log.println("Server started on " + d.toGMTString() + " ("
+ Time.ymdhms(d, true) + " server's local time)\n" );
log.flush();

try {
listen_socket = new ServerSocket(port);
}
catch (IOException e) {
fail(e, "Exception creating server socket");
}

participants = new Hashtable();

ticker = new Ticker(REAPER_PERIOD, this);

this.start();
}

public void run() {
try {
while(true) {
Socket client_socket = listen_socket.accept();
Connection c = new Connection(client_socket, this);
}
}
catch (IOException e) {
fail(e, "Exception while listening for connections");
}
}


public static void fail(Exception e, String msg) {
System.err.println(msg + ": " + e);
System.exit(1);
}


public void tick() {
// All the connections that have received no data for longer
// than the timeout shall be killed.
Vector toBeKilled = new Vector();
Date d = new Date();
long now = d.getTime();
Enumeration e = participants.keys();
while (e.hasMoreElements()) {
String key = (String) e.nextElement();
Connection thisClient = (Connection) participants.get(key);
long inactivity = now - thisClient.timeOfLastActivity;
if (inactivity > maxInactivity) {
toBeKilled.addElement(thisClient);
} else if (inactivity > warningIfInactiveFor) {
thisClient.out.println("INACTIVITY WARNING: you'll be kicked off in "
+ ((maxInactivity - inactivity)/1000) + " seconds");
thisClient.out.flush();
}
}
// now we've finished with participants, we can safely
// do things that remove stuff from it.


for (int i= 0; i < toBeKilled.size(); i++) {
Connection thisClient = (Connection) toBeKilled.elementAt(i);
// TODO: broadcast that we're throwing him out for inactivity
//System.err.println ("Killing " + thisClient.clientName);
try {
thisClient.client.close();
//System.err.println ("Client killed ok");
}
catch (IOException e2) {
fail (e2, "Couldn't kill client");
}
}
}

}




// For every chat client that logs in, there is a new connection
thread.
class Connection extends Thread {
protected Socket client;
protected DataInputStream in;
protected PrintStream out;
protected InetAddress clientAddress;
protected int clientPort;
protected String clientName;
protected long timeOfLastActivity;

protected Hashtable participants;
protected PrintStream log;

public static void fail(Exception e, String msg) {
System.err.println(msg + ": " + e);
System.exit(1);
}

// Initialize the streams and start the thread
public Connection(Socket client_socket, JavaTalkServer server) {
logActivity("Accepted incoming connection, spawned thread.");
client = client_socket;
timeOfLastActivity = (new Date()).getTime();
participants = server.participants;
log = server.log;
```

```
try {
in = new DataInputStream(client.getInputStream());
logActivity("Successfully created input stream for new socket");
out = new PrintStream(client.getOutputStream());
logActivity("Successfully created output stream for new socket");
}
catch (IOException e) {
try {
client.close();
logActivity("Successfully closed client socket");
}
catch (IOException e2) {
}
logException(e, "Exception while getting socket streams");
//??? how to you kill this thread before having started it?
this.stop();
return;
}


clientAddress = client.getInetAddress();
clientPort = client.getPort();
clientName = clientAddress + ":" + clientPort;
participants.put(clientName, this);
logActivity("Got a connection from " + clientName);
out.println(server.about);
broadcast ("User " + clientName + " joined the conference.");
logPrint(whoString());
this.start();
}


public String whoString() {
String result ="\n CURRENTLY CONNECTED:\n";
Enumeration e = participants.keys();
while (e.hasMoreElements()) {
String key = (String) e.nextElement();
Connection thisClient = (Connection) participants.get(key);
result += " " + thisClient.clientName
+ " (from " + key + ")\n";
}
return result;
}

public void run() {
String line;
int len;
try {
for(;;) {
// read in a line
line = in.readLine();
logActivity("Received line: " + line);
timeOfLastActivity = (new Date()).getTime();
if (line == null) {
logActivity ("Received empty line: closing down");
break;
}
// command MYNAME changes the user's name
if (line.startsWith("MYNAME ")) {
String msg;
msg = "RENAME: " + clientName + " --> "
+ line.substring(7) + " ("
+ clientAddress + ":" + clientPort + ")";
clientName = line.substring(7);
broadcast(msg);
logPrint(whoString());
continue;
}

// command WHO lists who is connected
if (line.startsWith("WHO")) {
out.println (whoString());
out.flush();
continue;
}

// no command, just text.
broadcast (clientName + "> " + line);
}
}
catch (IOException e) {
logException (e, "Exception in main loop talking to "
+ clientName + "(" + clientAddress
+ ":" + clientPort + ")");
}
finally {
logActivity ("Connection to "
+ clientAddress + ":" + clientPort
+ " not working, I'll close it");
try {
client.close();
logActivity ("Successfully closed broken socket for "
+ clientAddress + ":" + clientPort);
}
catch (IOException e2) {
logException (e2, "Exception while closing socket for "
+ clientAddress + ":" + clientPort);
}
finally {
client = null;
logActivity("Broken socket forgotten");
participants.remove(clientAddress + ":" + clientPort);
broadcast ("User " + clientName + " ("
+ clientAddress + ":" + clientPort
+ ") left the conference.");
logPrint(whoString());
}
}
}

public void logException (Exception e, String message) {
System.err.println(message + ": " + e);
}
```

```
public void logActivity (String message) {
System.out.println(message);
}

public void logPrint (String message) {
Date d = new Date();
log.print("["+ Time.ymdhms(true) + "] " + message);
log.flush();
}

public void broadcast (String s) {
Enumeration e = participants.keys();
while (e.hasMoreElements()) {
String key = (String) e.nextElement();
Connection thisClient = (Connection) participants.get(key);
thisClient.out.println(s);
thisClient.out.flush();
}
logActivity ("Broadcasting line: " + s);
logPrint(s + "\n");
}
}


class Ticker extends Thread {
public int period; // period between ticks in milliseconds
protected Trigger trigger;

public Ticker (int period, Trigger trigger) {
this.period = period;
this.trigger = trigger;
this.start();
}

public void run() {
for (;;) {
if (trigger != null) {
trigger.tick();
}
try {
sleep(period);
} catch (InterruptedException e) {
System.err.println("Ticker thread interrupted: " + e);
}
}
}
}


interface Trigger {
// Something that can be called by a ticker to make things happen.
public abstract void tick();
}
```

```
class Time {
/*
NOTE: the year-month-day representation is a Good Thing (tm)
because it puts the most significant digits first.
So you can sort ymd strings alphabetically and they will
be sorted chronologically for free.
*/

/**
Returns a string with a ymdhms representation of the supplied
date, with or without lots of separators between the fields.
The separators make the string more readable but get in the
way if you are composing a file name.
*/
public static String ymdhms(Date d, boolean separators) {
int y = d.getYear();
int month = d.getMonth()+1;
int date = d.getDate();
int h = d.getHours();
int m = d.getMinutes();
int s = d.getSeconds();
String result = "";
if (y < 10) {
result += "0";
}
result += y;
if (month < 10) {
result += 0;
}
result += month;
if (date < 10) {
result += "0";
}
result += date + "-";
if (h < 10) {
result += "0";
}
result += h;
if (separators) {
result += ":";
}
if (m < 10) {
result += 0;
}
result += m;
if (separators) {
result += ":";
}
if (s < 10) {
result += "0";
}
result += s;
return result;
}


/**
Returns a string with a ymdhms representation of "now".
*/
public static String ymdhms(boolean separators) {
return ymdhms(new Date(), separators);
}


public static String hoursMins(Date d) {
int h = d.getHours();
int m = d.getMinutes();
String result = "";
if (h < 10) {
result += "0";
}
result += h + ":";
if (m < 10) {
result += 0;
}
result += m;
return result;
}

public static String hoursMins() {
return hoursMins(new Date());
}

}
```

JavaTalkClient Source Code

```
// $RCSfile: JavaTalkClient.java,v $
//
// $Id: JavaTalkClient.java,v 1.25 1996/07/15 20:19:18 fms Exp fms $
//
// by Frank Stajano, http://www.cam-orl.co.uk/~fms
// Development started on 1996 05 09
// (c) Olivetti Research Limited

/*
Copyright (c) 1996 Olivetti Research Limited

Permission is hereby granted, without written agreement and
without license or royalty fees, to use, copy, modify, and
distribute this software and its documentation for any purpose,
provided that the above copyright notice and the following three
paragraphs appear in all copies of this software, its documentation
and any derivative work.

In no event shall Olivetti Research Limited be liable to any party
for direct, indirect, special, incidental, or consequential damages
arising out of the use of this software and its documentation, even
if Olivetti Research Limited has been advised of the possibility of
such damage.

Olivetti Research Limited specifically disclaims any warranties,
including, but not limited to, the implied warranties of
merchantability and fitness for a particular purpose. The
software provided hereunder is on an "as is" basis, and Olivetti
Research Limited has no obligation to provide maintenance,
support, updates, enhancements, or modifications.

Derived or altered versions must be plainly marked as such, and
must not be misrepresented as being the original software.
*/


// TRIVIAL TODO
// only honour enter key if focus is in small window
// provide buttons for functions like MYNAME and WHO
// make font and size changeable by user
// play cute sound when someone joins

// ADVANCED TODO
// implement the more complete timestamp protocol
// make window floating
// provide separate window for roundtrip messages
// synchronize server and client clocks

import java.io.*;
import java.net.*;
import java.applet.*;
import java.awt.*;
import java.util.*;
```

```
public class JavaTalkClient extends Applet {
// These are automatically substituted
public static final String rcsVersion = "$Revision: 1.25 $";
public static final String rcsDate = "$Date: 1996/07/15 20:19:18 $";
public static final String rcsProgram = "$RCSfile:
JavaTalkClient.java,v $";

// These extract the useful parts from the automatically substituted
ones.
public static final String version = rcsVersion.substring(11,
rcsVersion.length()-2);
public static final String date = rcsDate.substring(7, 17);
public static final String program = rcsProgram.substring(10,
rcsProgram.length()-9);
public static final String about = "";

public static final int DEFAULT_PORT = 6764;
public static final String DEFAULT_HOST = "cyclonus.mit.edu";
public static final boolean DEFAULT_SHOWROUNDTRIP = false;
public static final boolean DEFAULT_SHOWTIMESTAMPS = false;

public static final int MAX_CHARS = 10000;
// When the text window contains this
// amount of stuff or more,
// it's time to start chopping old stuff off the top.
// NB: weird behaviour (i.e. you can't write to the window
// any more) is observed around the 27000-28000
// characters mark. I don't know what imposes the limit and
// haven't found any documentation on it yet-- I
// thought it would be more like 64kb, i.e. 32k unicode chars.
// This constant used to be set at 32000 and the program would
// freeze when reaching 28000 or so...

public static Random idGen = new Random();
// random number generator, common to the class so that successive
// applets use the same stream instead of restarting a new generator

protected Listener listener = null;
protected Typist typist = null;
protected int port;
protected String host;
protected int maxChars;
protected Socket s = null;
protected String myAddressAndPort;
protected int myId = idGen.nextInt();
protected boolean showRoundTripInitially;
protected boolean showTimestampsInitially;

public String[][] getParameterInfo() {
String[][] info = {
// name, type, description
{"host", "string", "internet name of the host on which the server
runs"},
{"port", "integer", "port on which the server is listening for
connections"},
{"showRoundTrip", "boolean", "initial value of corresponding
checkbox"},
{"showTimestamps", "boolean", "initial value of corresponding
checkbox"},
{"maxChars", "integer", "maximum number of characters in window before
top chopped off"}
};
return info;
}

public String getAppletInfo() {
return about;
}

public void init() {
// Figure out parameters
System.err.println("init() called");
System.err.println(about);

try {
port = Integer.parseInt(getParameter("port"));
}
catch (NumberFormatException e) {
port = DEFAULT_PORT;
}

host = getParameter("host");
if (host == null) {
host = DEFAULT_HOST;
}
showRoundTripInitially = Boolean.valueOf(
getParameter("showRoundTrip")).booleanValue();
showTimestampsInitially = Boolean.valueOf(
getParameter("showTimestamps")).booleanValue();

try {
maxChars = Integer.parseInt(getParameter("maxChars"));
}
catch (NumberFormatException e) {
maxChars = MAX_CHARS;
}


System.out.println("host = " + host
+ "\nport = " + port
+ "\nshowRoundTrip = " + showRoundTripInitially
+ "\nshowTimestamps = " + showTimestampsInitially
+ "\nmaxChars = " + maxChars
);
}

public void destroy() {
System.err.println("destroy() called");
}

public void finalize() throws Throwable {
System.err.println("finalize() called");
```

```
}

public void start() {
System.err.println("start() called");
// make socket, windows and threads
try {
s = new Socket(host, port);

myAddressAndPort = s.getInetAddress() + ":"+ s.getPort();
System.err.println("Connected to server at " + myAddressAndPort);
}
catch (IOException e) {
System.err.println("Couldn't make socket: "+ e);
this.cleanUp();
return;
}


//{{INIT_CONTROLS
setLayout(null);
resize(511,377);
winLocal=new TextField(44);
winLocal.setFont(new Font("Courier",Font.PLAIN,10));
add(winLocal);
winLocal.reshape(126,315,371,30);
typePrompt=new Label("Type here:", Label.RIGHT);
typePrompt.setFont(new Font("Helvetica",Font.PLAIN,12));
add(typePrompt);
typePrompt.reshape(7,330,112,15);
winAll=new TextArea(19,60);
winAll.setFont(new Font("Courier",Font.PLAIN,10));
add(winAll);
winAll.reshape(7,0,497,308);
showRoundTripCheck=new Checkbox("Show round trip time");
add(showRoundTripCheck);
showRoundTripCheck.reshape(126,353,154,22);
showTimestampsCheck=new Checkbox("Show timestamp messages");
add(showTimestampsCheck);
showTimestampsCheck.reshape(315,353,182,22);
//}}

showRoundTripCheck.setState(showRoundTripInitially);
showTimestampsCheck.setState(showTimestampsInitially);
winAll.setEditable(false);
winAll.appendText(about + "\n");
listener = new Listener(s, winAll, this);
typist = new Typist(s, winLocal, this);

}

public void stop() {
System.err.println("stop() called");
cleanUp();
super.stop();
}

public void cleanUp() {
```

```
System.err.println("cleanUp() called");

// remember these in case the user stops and restarts
showRoundTripInitially = showRoundTripCheck.getState();
showTimestampsInitially = showTimestampsCheck.getState();

// stop threads, remove windows and delete socket
if (typist != null && typist.running) {
typist.stop();
}
typist = null;
System.err.println("typist thread stopped and forgotten");

if (listener != null && listener.running) {
System.err.println("about to stop listener...");
listener.stop();
System.err.println("listener stopped ok");
}
listener = null;
System.err.println("listener thread stopped and forgotten");

removeAll();
System.err.println("windows deleted");

try {
if (s != null) {
s.close();
System.err.println("socket closed ok");
}
}
catch (IOException e) {
System.err.println("Couldn't close socket: " + e);
}
s = null;
myAddressAndPort = "";
repaint();
}


public boolean keyUp(Event e, int key) {
// When the applet receives a key event, if it was a Return
// then it is dispatched to the typist thread.
int flags = e.modifiers;
if ((e.id == Event.KEY_RELEASE) && (e.key == '\n')) {
if (typist != null) {
typist.sendLine();
}
return true; // I handled it myself
} else {
return false; // I want someone else to handle it
}
}


public void paint(Graphics g) {
if (typist == null && listener == null && s == null) {
g.setFont(new Font("Helvetica",Font.BOLD,18));
g.drawString("Not connected to server. Threads stopped.", 20, 180);
}
}


public void safelyAppendTextToMainWindow(String toBeAppended) {
// trim top lines off the widget when we are near the max size
// or it will become full and it won't take any more.
TextArea w = winAll;
String text = w.getText();
while (text.length() + toBeAppended.length() > maxChars) {
int upTo = text.indexOf("\n");
w.replaceText("", 0, upTo+1);
text = w.getText();
}
w.appendText(toBeAppended);
}

//{{DECLARE_CONTROLS
TextField winLocal;
Label typePrompt;
TextArea winAll;
Checkbox showRoundTripCheck;
Checkbox showTimestampsCheck;
//}}

}



class Typist extends Thread {
protected PrintStream sout;
protected TextField w;
protected boolean running = false;
protected JavaTalkClient applet;
protected long lineId = 0;

public Typist(Socket s, TextField window, JavaTalkClient applet) {
System.err.println("Making a typist...");
w = window;
this.applet = applet;
try {
sout = new PrintStream(s.getOutputStream());
System.err.println("Typist thread opened output stream to server");
}
catch (IOException e) {
System.err.println ("Typist thread can't open output stream to
server");
running = false;
applet.cleanUp();
return;
}
this.start();
}

public void run() {
System.err.println("Typist thread running...");
```

```
running = true;
}

protected void finalize() throws Throwable {
sout.close();
sout = null;
System.err.println("Typist thread closed its output stream.");
System.out.println("Typist thread stopped.");
}


public void sendLine() {
String line = w.getText();
w.setText("");

if (line == null) {
}

line = line.trim() + "\n";

if (applet.showRoundTripCheck.getState()) {
long timeSent = (new Date()).getTime();
sout.print("TIMESTAMP myid:" + applet.myId
+ " lineid:" + lineId
+ " timesent:" + timeSent + "\n");
}
sout.print(line);
sout.flush();

lineId++;
}
}


class Listener extends Thread {
protected DataInputStream sin;
protected TextArea w;
protected boolean running = false;
protected JavaTalkClient applet;

public Listener(Socket s, TextArea window, JavaTalkClient applet) {
System.out.println("Making a listener...");
w = window;
this.applet = applet;
try {
sin = new DataInputStream(s.getInputStream());
System.out.println("Listener thread opened its input stream.");
}
catch (IOException e) {
System.err.println ("Listener thread can't open input stream: " + e);
running = false;
applet.cleanUp();
return;
}
this.start();
}
```

```
public void run() {
// listens to the socket and, whenever the server says something,
// puts it in the window.

System.out.println("Listener thread starting...");
running = true;

String line;
while (true) {
try {
line = sin.readLine();
}
catch (IOException e) {
System.err.println("Listener: error in readLine: " + e);
running = false;
applet.cleanUp();
return;
}

// Check if connection is closed (i.e. for EOF)
if (line == null) {
System.out.println("Listener got EOF from server: closing");
running = false;
applet.cleanUp();
return;
}

String roundTripPrompt = "";
if (applet.showRoundTripCheck.getState()) {
Date date = new Date();
int match = line.indexOf("TIMESTAMP myid:"
+ applet.myId);
if (line.indexOf("TIMESTAMP myid:"
+ applet.myId) != -1) {
int i = line.indexOf("lineid:");
int j = line.indexOf("timesent:");
String lineIdString =
line.substring(i+"lineid:".length(), j).trim();
int lineId = Integer.parseInt(lineIdString);
long timeSent = Long.parseLong(
line.substring(j+"timesent:".length()));
long timeReceived = date.getTime();
long roundTrip = timeReceived - timeSent;
roundTripPrompt = "[roundtrip: " + roundTrip + " ms] ";
}
}

// Show timestamp lines only on request,
// but show other lines all the time.
//
// NB: do not confuse "timestamp lines" with "roundtrip prompt"
// because they are different! The timestamp line comes in from
// the server, while the roundtrip prompt is generated locally.
// A timestamp line plus a message line take 2 passes round the loop:
// after each pass, the roundTripPrompt is printed, but it will
// be empty unless the line of the previous pass was a timestamp.
if (applet.showTimestampsCheck.getState()
|| (line.indexOf("TIMESTAMP myid:") == -1)) {
applet.safelyAppendTextToMainWindow (line + "\n");
}

applet.safelyAppendTextToMainWindow(roundTripPrompt);

}
}
}
```

JavaTalk Readme File

Olivetti JavaTalk
=================
1996 07 16

JavaTalkClient v. 1.25
JavaTalkServer v. 1.17

INDEX:
Introduction
Licence and copyright
What do you need?
Connecting as a user
Running a JavaTalk server
Acknowledgements
Packing list
Contact information

Introduction

JavaTalk is a simple but effective multiuser chat program. Someone somewhere runs the server software and installs a suitable web page, then everyone else can join the chat by simply going to that page with their browser. You go to that page and whatever you type appears on the screen of everybody else who is watching the page at the same time. As simple as that! No messing around, no software installation, no client software to run, no command set to learn. If you want to give it a test drive, come to

http://badges.cam-orl.co.uk/~fms/javatalk

Writing a chat system in Java has been a popular idea with many developers and if you search the web you'll certainly find many more implementations. I know of at least four more. So what has JavaTalk got for you that makes it better than the others? Well -- that's up to you to decide, but here are three basic reasons to consider.

* SIMPLE. No complicated command set to remember. No operators, chat rooms, privileges, private whispers or any of that stuff. Just go to the JavaTalk page and anything you type is seen by everyone else. Zero training needed.

* SMALL. The classes for the client software total less than 15K. That means quick loading time, even on slow links.

* FREE. Because I've got a smart boss, both the executable and the source are available to you for free, no licence form to fill in, no questions asked. You can use it as you wish on your own web pages, or as an internal communication facility in your intranet.

Licence and copyright

What do you need?

* To participate as a client:

You need a Java-enabled browser. Then simply connect to a page that runs a JavaTalk server (like for example the JavaTalk home page mentioned above) and tell your friends to join you there at the arranged time.

You don't need to download anything, neither the executable nor the source release of JavaTalk. Everything you need will be picked up automatically by your browser as and when you visit the web page.

* To run your own JavaTalk server:

You need the ability to run a Java interpreter on the same machine as your web server.

Grab the executable release, unpack it in its own directory, edit the example web page if you want to, and launch the server.

* To modify the program:

You need a Java compiler and developer's kit. Currently available for free from Sun, as well as for money from commercial vendors.

Grab the source release.

All of the above, plus info on newer versions if any, can be found on the JavaTalk home page (see "contact info" at the end of this file).

Connecting as a user

Most of what appears in this section should be easy to figure out without instructions, but I'll spell it out in detail just so that you have a reference handy.

Go to the page of the JavaTalk server you want to connect to. The fact of going to the page makes your browser download the applet and connects you to the JavaTalk server.

When the applet has loaded you'll see two windows: a big one above and a one-line one below. You type, one line at a time, in the one-line window, pressing Enter after each line. What you type is sent to the server which then echoes it back to the upper window of everyone who is connected.

The echoed lines are prefixed with the name of the sender. You can set your name to whatever you want by using the command "MYNAME xxx" (commands are all uppercase so that they are unlikely to clash with anything you normally type). Replace xxx with your chosen name, of course. Don't choose a terribly long name or there won't be much room left for your messages and people will have to scroll horizontally, which is a nuisance.

Before you use MYNAME for the first time, your default name will be the internet address and port number of the socket you are using to connect to the server.

To see who else is connected to the server at the time, type "WHO" (the second and last command understood by the current version of the server).

If you don't type anything for a while, the server will suspect that you went away and will warn you. If you want to stay in, type something (and press Enter to send it). Otherwise, after another little while, you'll be disconnected.

(Technical note: this reaping process is performed to get rid of those zombie users who used to appear as connected even though they had long since gone away. It turns out that a known bug (SUN bug 1234731, see http://java.sun.com/java.sun.com/products/JDK/1.0.2/KnownBugs.html) in the Windows implementation of the Java socket libraries prevented the JavaTalk server from detecting that they had gone away. So I pre-empted the problem by kicking out inactive users, regardless of whether their socket looked OK or not. If you run your own JavaTalk server, you can customise the timeouts.)

When you are disconnected from the server, either because the server kicked you out or because the server couldn't be contacted on the given address and port (maybe because it wasn't running?) then the applet will not display any windows or buttons: it will just say "Not connected to server. Threads stopped." To reconnect (assuming the server is there to connect to), just reload the page.

Every time you leave the page, you disconnect from the server (though of course you won't see the above message because the applet won't be visible any more). The other participants to the conference will get the message "User xxx left the conference". Conversely, every time you enter the page, you join the conference and again the others are notified.

There are two checkboxes at the bottom of the applet that activate a debugging feature to measure the time it takes for your messages to travel from you to the server and back (the roundtrip delay). This is done by sending a special extra timestamp line every time you send a regular line. The server doesn't do anything special-- it just echoes the timestamp like it does with every other line. Your client, though, recognises it when it gets it and with some advanced mathematics (namely a subtraction) it computes how long the message took to get back. This time is then printed in the prompt that precedes the actual message. All of this stuff is activated and deactivated with the first checkbutton, the one that says "Show round trip time". The other one ("Show timestamp messages") is rather useless unless you are me and are trying to debug the timestamping protocol: it deactivates or reactivates the filtering of the special timestamps. It probably shouldn't even be there in a production release, and may disappear without notice in the future.

Running a JavaTalk server

Note: throughout this section it is assumed that you have already installed a Java environment on your system, and in particular that the Java interpreter and the applet viewer are available on the path. If this is not the case, do that first. (See http://www.javasoft.com for details and software.)

Because of the restrictions imposed by Java's security model, an applet can only make a TCP connection to the machine it was loaded from. This means that the JavaTalk server must necessarily run from the same machine as the web server that is serving the html page in which the JavaTalk client applet is embedded.

For a quick start, make a JavaTalk subdirectory in your public html directory and unpack the executable distribution in it.

For a test run, start the server with the default parameters (on my systems I type "java JavaTalkServer") and, from another console on the same machine, run the client inside the appletviewer (on my systems I type "appletviewer index.html"). You should be able to type in the bottom window of the client and see your messages echoed (by the server) in the top window. Of course you can start more copies of appletviewer and they will chat to each other.

To try out JavaTalk with your web browser instead of the appletviewer, you'll have to edit the index.html page. Change the <param name=host value=localhost> entry, replacing localhost with the fully qualified Internet name of the machine where the JavaTalk server is running (for example badges.cam-orl.co.uk). Then, ensuring that this page is served by your web server (check the file permissions), you can point your browser at it and you should see the applet embedded in the page.

Once you've checked that the supplied page works, you can start modifying its text and layout to suit your purposes. It is appreciated, but not required, that you keep the JavaTalk logo and the link to the JavaTalk home page if you use the unmodified server and client from the executable release. It is required that you do NOT use the JavaTalk logo if you run your own modified versions of the programs. You should instead say something like "derived from Olivetti JavaTalk (hyperlink to the genuine thing) with unapproved modifications".

As the server runs, it keeps a transcription of the talk in a log file. Every time the server starts, it opens a new log file, named after the current date and time and so very likely to be unique. Killing and relaunching the server is the approved way to start a new log file.

Note: the log will not contain all the messages that the server normally prints on the console. If you want the server to shut up (but still write to the log as usual), redirect its standard output and standard error.

You can supply your own log file name if you want with a command line parameter. If you give the name of an existing file, the file will be overwritten, not appended to. You can give /dev/null, or its equivalent on non-unix systems, if you don't want a log file to be produced.

While we are on that subject, here are the parameters accepted by the server, with their defaults: they are all optional. Note that the server prints out a help message if you invoke it with "help", "-help", "-h" and so on (actually, anything that's not a valid integer) as the first parameter.

```
USAGE (all arguments optional and positional):
java JavaTalkServer warning afterWarning port logFileName

warning
(default 300)
After this many seconds of inactivity, the user gets a warning.

afterWarning
(default 60)
After this many extra seconds of inactivity after the first warning,
the user is kicked out.
```

port
(default 6764)
Port number on which to listen for connections. If you run more than one server on the same machine, each one must listen on a different port.

logFileName
(default javatalk.YYMMDD-HHMMSS.log, with suitable replacements)
A time-stamped transcript of everything the participants say during the lifetime of the server goes here.

Because the arguments are positional, you can only omit arguments from the tail end. For example, if you only want to specify a new port number of 7000, you still have to give some value for warning and afterWarning, as in

```
java JavaTalkServer 300 60 7000
```

It is accepted JavaTalk etiquette to tell your users, on the page with the client, that all the chats are logged.

For pre-organised meetings, it is also accepted JavaTalk etiquette to send a copy of the log to all the participants.

If you are editing the web page that contains the applet, you'll want to know the parameters that the applet takes: they are listed below, and they are shown in the example page.

port
(default 6764)
Port number to connect to. Please ensure that it matches the port number that the server is listening on. You'll have to change this value if you use many servers to provide several chat rooms.

host
(default badges.cam-orl.co.uk)
Fully qualified name of the host that the server is running on. You'll definitely want to change this one.

showRoundTrip
(default false)
Initial state of the "Show round trip" checkbox.

showTimestamps
(default false)
Initial state of the "Show timestamps" checkbox.

maxChars
(default 10000)
Maximum number of characters that will be accepted in the scrolling window before we start chopping old stuff off the top. Don't set it too high or the widget will fill and lock up. (NB: 32K minus epsilon is still too high. Experimentally,

it locks up at about 27000, but I like to err on the side of safety.)

JavaTalk doesn't have multiple "rooms" or "subconferences", but you can implement this by dedicating a different web page to each. Of course, behind each page must be a different instance of the JavaTalk server, listening on its own dedicated port. This means that the various pages will have to supply different values for the "port" parameter.

The disadvantage of this scheme is that people can't open new conferences dynamically. But the advantage (other than simplicity) is that the conferences are completely decoupled and you can even safely shut down the server for one of the conferences without affecting any of the others that are running.

Acknowledgements

The working core of JavaTalk was written in two days of furious hacking (9 & 10 May 1996). Alessandro Morales gets credit as the first beta-tester: I dragged him in with an international phone call on the evening of the 10th, as soon as I got the prototype to work. Quentin Stafford-Fraser was always available to discuss Java issues and contributed with helpful advice on many occasions. Adriano Vernassa helped with stress-testing of the applet over poor routing conditions (the roundtrip-measuring code went in because of the problems he was experiencing). Hiang-Swee Chiang was kind enough to read through my code before the public release and he came back with some helpful comments. Harold Syfrig helped me track down a scrolling-related bug in the client. My thanks to all these people, though of course any bugs and shortcomings remain my responsibility.

Packing list

Note: you'll need an unzip program capable of dealing with long filenames. There are free versions of such a program both for Unix and for Win32. Alternatively you'll need a tar and gunzip pair, again capable of long file names. Again, there are free versions of such programs for both Unix and Win32.

I keep wondering why I can't get away with supplying only one flavour, but then I'm a nice guy and I want to make your life easier, so please pick the compression flavour that you're already familiar with. After going to the trouble of providing two sets of distributions, I decided to make the .tar.gz one more unixy (LF) and the .zip one more windowsy (CRLF). This only affects the line terminators in the text files. Everything else is exactly the same.

Executable release:

```
JavaTalk-executable.zip or JavaTalk-executable.tar.gz

./readme.txt this file
./client/index.html sample web page containing the applet
./client/javatalk.gif JavaTalk logo appearing on page
./client/*.class classes implementing the client applet
./server/*.class classes implementing the server


Source release:

JavaTalk-source.zip or JavaTalk-source.tar.gz

./readme.txt this file
./JavaTalkClient.Java source for the client
./JavaTalkServer.Java source for the server


In case you're wondering, I developed the software with Symantec Cafe'
under Windows 95. I normally run the server on Unix (Solaris). I tend
to run the client on Windows via Netscape but I occasionally run it
from Unix too (Solaris and OSF, again with Netscape).



Contact information
-------------------

I welcome your feedback. If you want to be kind, start your subject
line with "javatalk" so that the message will go in the right mail
folder by itself.

Frank Stajano (filologo disneyano)

Olivetti Research Limited
24A Trumpington Street
Cambridge CB2 1QA
UK

http://www.cam-orl.co.uk/~fms
mailto:fstajano@cam-orl.co.uk

The JavaTalk page: http://badges.cam-orl.co.uk/~fms/javatalk

--- END of readme.txt ---
```

JavaTalk Example Web Page

```
<html>

<head>
<title>JavaTalk 1.1</title>
</head>

<body>
<!-- Insert HTML here -->

<p align="center">&nbsp;</p>

<p align="center">
<applet code="JavaTalk.class" width="415" height="300">
  <param name="host" value="cyclonus.mit.edu">
  <param name="port" value="600">
</applet>
</p>
</body>
</html>
```

Viewing Room Prototype Page

```
<html>

<head>
<title>JavaTalk 1.1</title>
</head>

<body>
<!-- Insert HTML here -->

<p align="center">&nbsp;</p>

<p align="center"><embed width="128" height="128" src="Episode312.ram"></p>

<p align="center">
<applet code="JavaTalk.class" width="415" height="300">
  <param name="host" value="cyclonus.mit.edu">
  <param name="port" value="600">
</applet>
</p>
</body>
</html>
```

From: "Lori Robertson (ERA)" <Lori.Robertson@era.ericsson.se>
To: deco@media.mit.edu, gid@media.mit.edu
Cc: "Fiona Williams (EED)" <Fiona.Williams@eed.ericsson.se>
Subject: RE: [Fwd: Media Lab Notes on key points]
Date: Wed, 24 May 2000 14:34:17 +0200
X-Mailer: Internet Mail Service (5.5.2651.58)
X-OriginalArrivalTime: 24 May 2000 12:32:17.0953 (UTC) FILETIME=
[193A4910:01BFC57C]

Dear Gloriana,
I would like to try to have a short telephone conference with you prior to the 31st of May. I have been
speaking with the 2 potential fellowships from our lab regarding the possibility of doing their studies in conjunction with MediaLabEurope. I have a number of practical questions regarding how
to proceed with their continued studies.

Both candidates could consider doing their doctorate courses at a University in Ireland such
as Trinity College - Computer Vision & Robotics Group. We are currently investigating universities that
have similar programs of interest.

Could you possibly contact me at + 46 8 404 6994? I can also be reached at + 46 70 5816354.

Best Regards,

Lori Robertson

-----Original Message-----
From: Terry Turner [mailto:tt@broadcom.ie]
Sent: den 17 maj 2000 16:08
To: deco@media.mit.edu; gid@media.mit.edu
Cc: fiona.williams@eed.ericsson.se; lori.robertson@era.ericsson.se
Subject: [Fwd: Media Lab Notes on key points]

Hi Deb and Glorianna,

On behalf of Fiona, attached is a file produced directly after the
meeting today.

Best regards,

Terry

--
Terry Turner
R&D Operations Manager
Broadcom Eireann Res. Ltd.   Phone: +353 1 6046028
Kestrel House   Fax: +353 1 6761532
Clanwilliam Place
Dublin 2